

Sistemas Operativos

# Ayudantía 1: Fork

Profesores: Víctor Reyes, Yerko Ortíz

Ayudante: Diego Banda



# Contacto



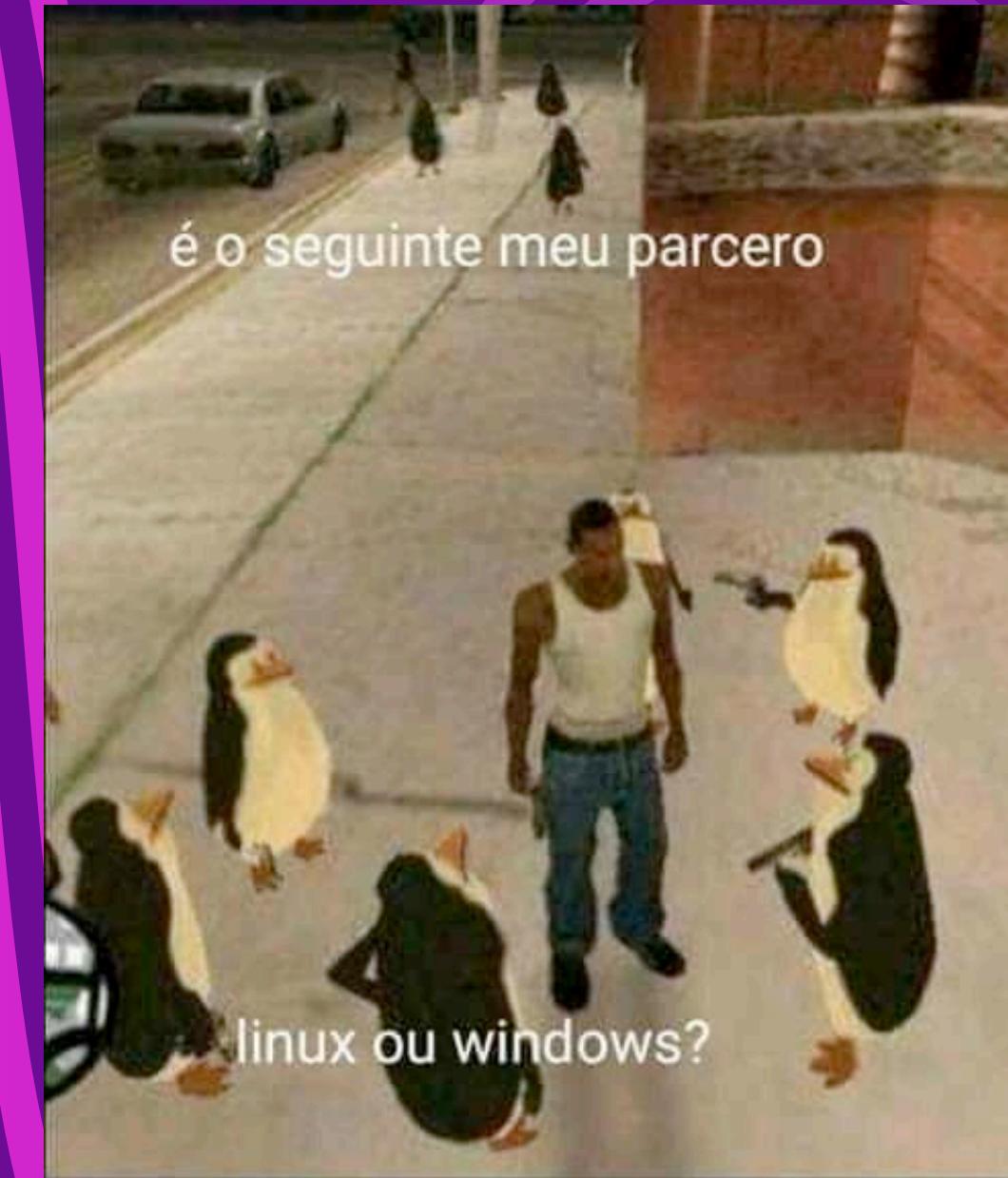
diego.banda@mail\_udp.cl



darklouds\_



<https://github.com/DiegoBan/SO-2025-2#>





**¡¡Regla muy  
importante imposible  
de olvidar!!**



Windows queda  
estrictamente  
prohibido en cualquier  
tipo de clase, ejercicio,  
tarea y cualquier cosa  
relacionada al curso

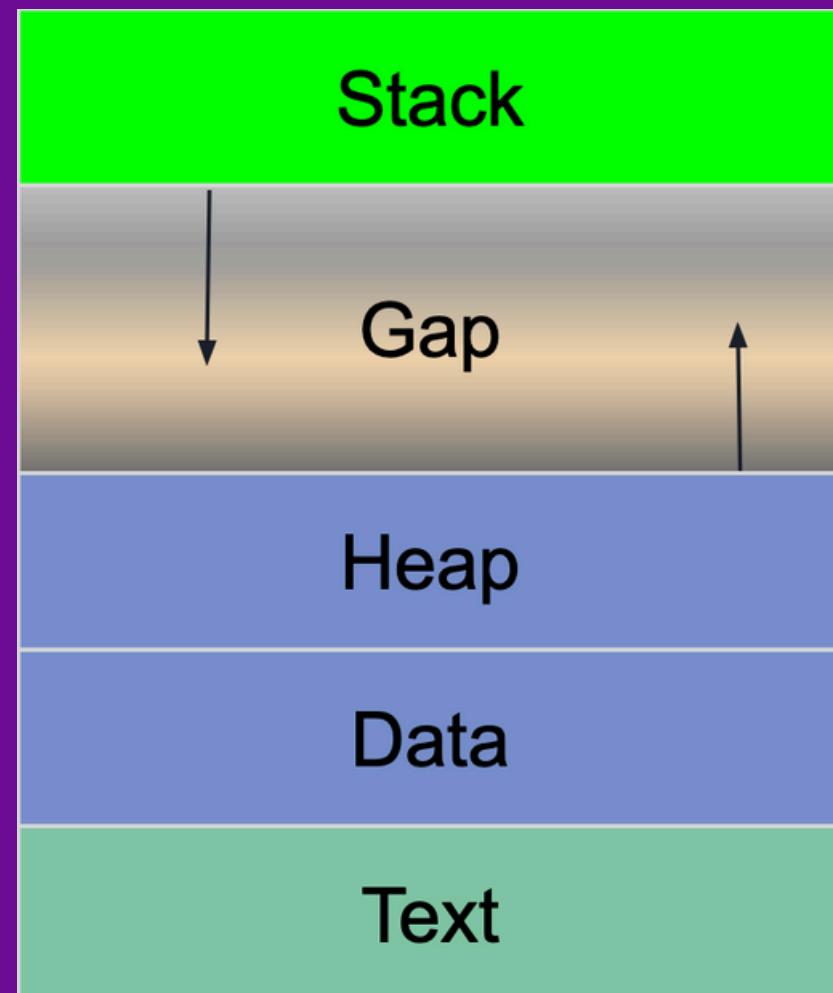


# Procesos

¿Qué es un proceso? Es absolutamente todo lo que existe dentro del computador, desde el SO hasta un proceso cualquiera. Existen muchos procesos a la vez en un computador, pero se tienen menos núcleos que procesos...

Estos procesos se “intercalan” a través del núcleo, para ello es necesario guardar información:

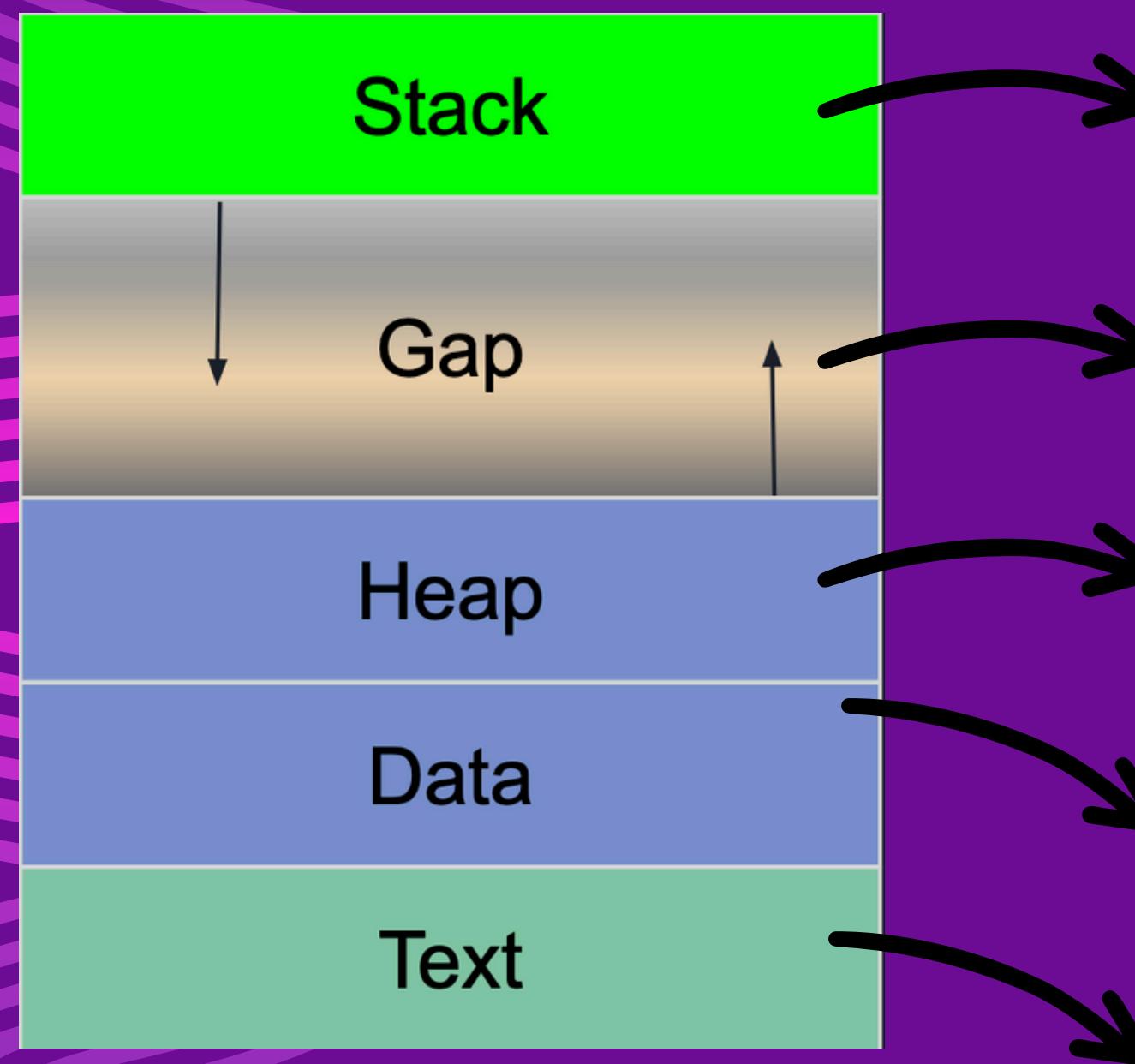
Proceso en memoria



|                            |
|----------------------------|
| Estado del proceso         |
| Número del proceso         |
| Program counter            |
| Límites de memoria         |
| Lista de archivos abiertos |
| ...                        |

Process Control Block (PCB)

# Procesos



Guarda las variables locales entre otros registros, crece hacia abajo.

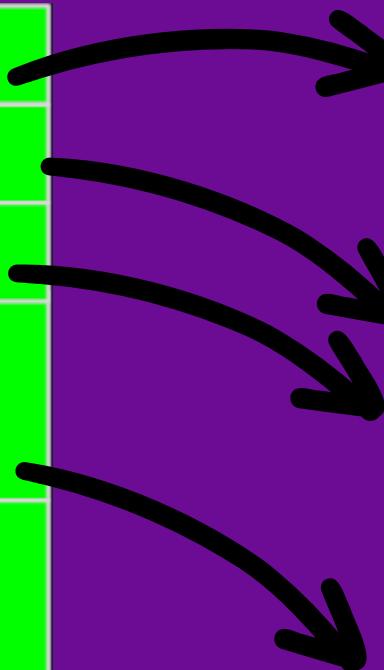
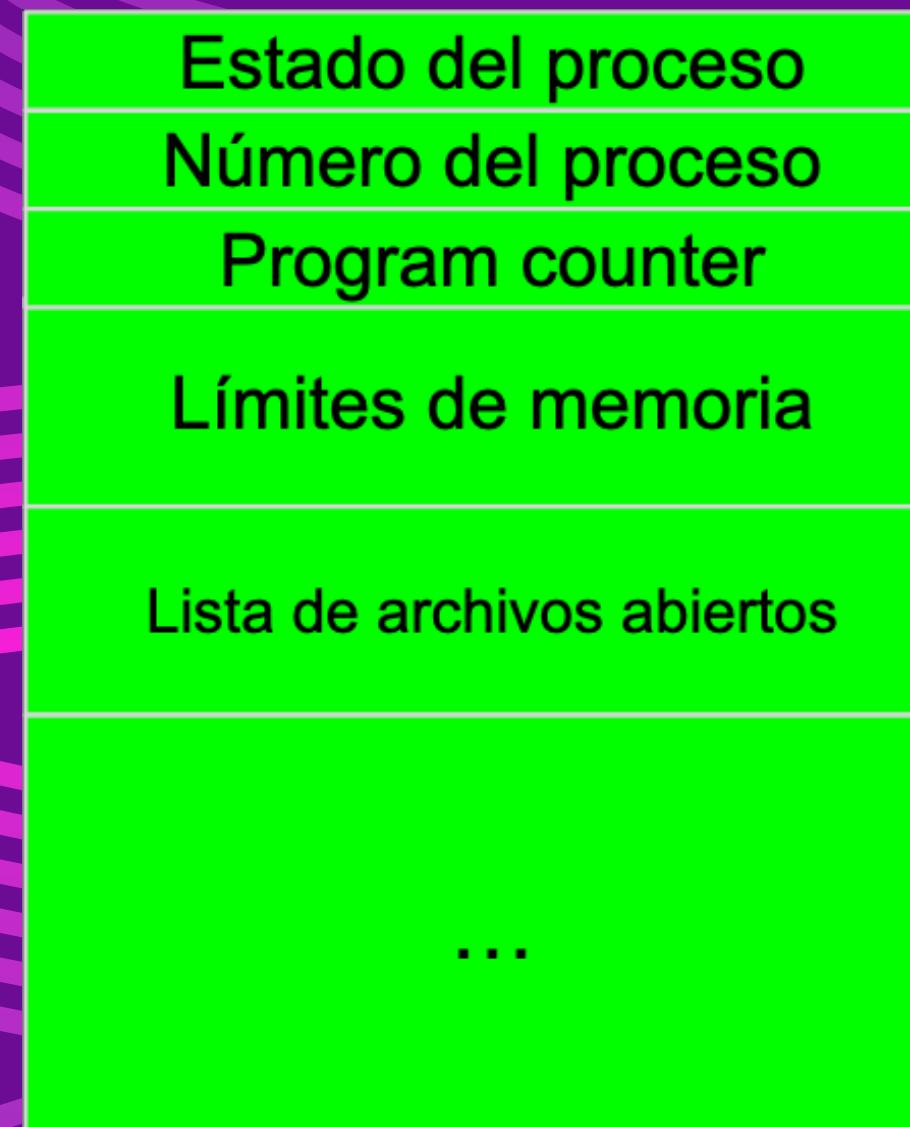
Espacio entre Stack y Heap hecho para crecer según lo necesario.

Se usa como memoria dinámica en tiempo de ejecución (para new en C++ por ejemplo, o malloc en C), crece hacia arriba.

Variables globales y/o estáticas ya inicializadas.

El código ejecutándose en el proceso.

# Procesos



Estado actual (running, ready, waiting, terminated).

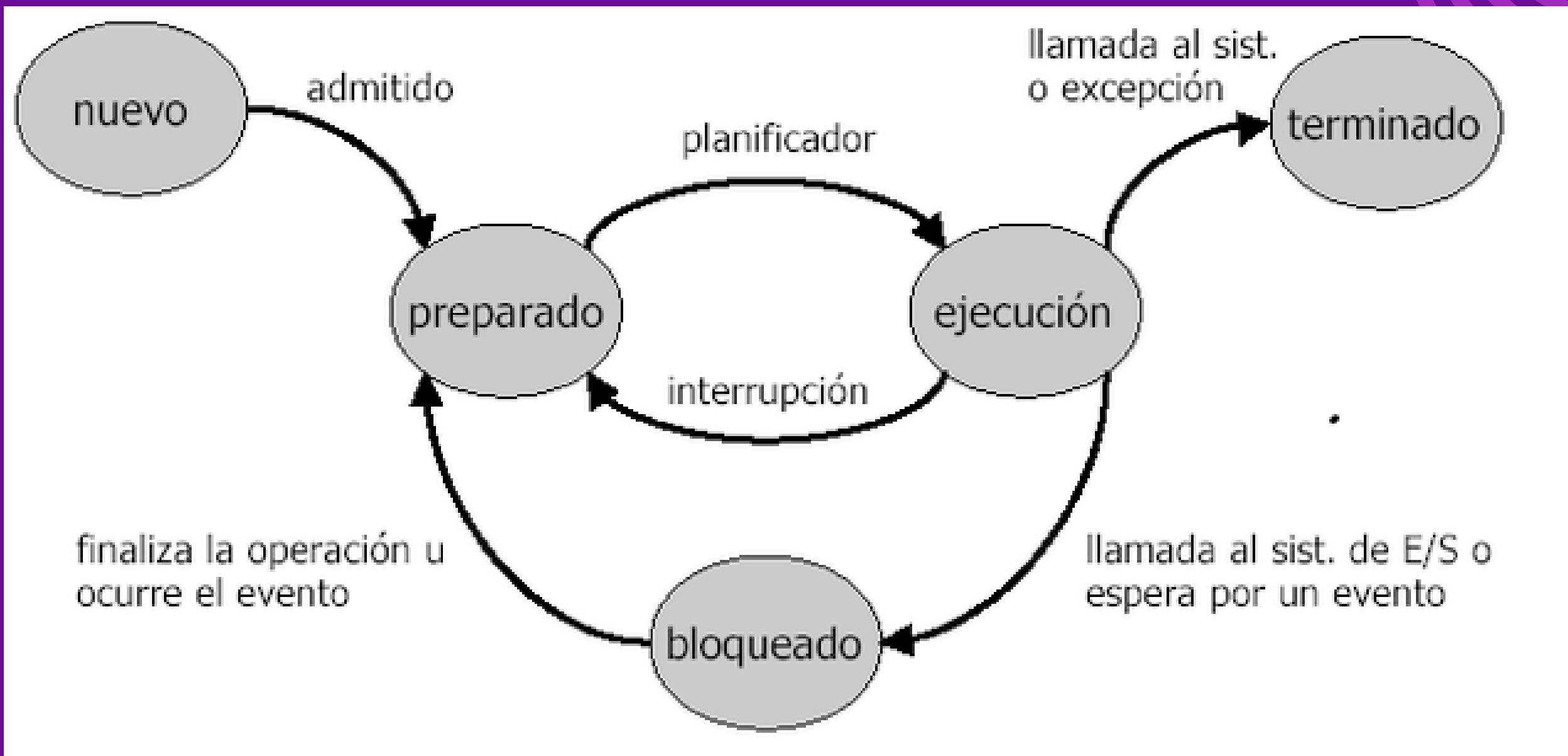
Identificación única Process ID (PID).

Dirección de la próxima instrucción a ejecutar.

Límites de memoria asignada al proceso.



# Procesos



# Syscalls

Enlace entre modo kernel y modo usuario: Permite al usuario hacer una acción que solo entrega el Sistema Operativo a través del acceso al kernel.

→ Pueden variar dependiendo del SO.

Algunas de las más importantes son:

- **fork()**: Bifurcación, crea una copia exacta del proceso (convirtiéndose en un proceso hijo del original), con las mismas funciones, variables, espacio en memoria, etc.
- **wait()**: Un proceso padre espera el término de uno hijo.
- **exit()**: Termina la ejecución del programa.
- **sleep()**: Suspende la ejecución del programa durante la duración determinada.
- **getpid()**: Retorna el Process ID del proceso actual.



# Procesos

**Proceso Huérfano:** Ocurre cuando el padre termina antes que su hijo y no lo espera con alguna instrucción cómo **wait()**. En este caso se le asigna al proceso init (PID = 1) el cual si espera que termine.



Pensó que era huérfano hasta el episodio 5



**Proceso Zombi:** Ocurre cuando un proceso hijo ya terminó su ejecución, pero el padre aún no lee el estado de termino o salida. Técnicamente ya terminó el proceso, pero sigue existiendo en la tabla de procesos para que el padre pueda leer su estado.



# Fork

Genera una ‘bifurcación’, creando un proceso hijo que es una copia del padre y retorna un pid con diferentes significados:

- **pid > 0:** Estamos en el proceso padre y el pid corresponde al del hijo
- **pid = 0:** Estamos en el proceso hijo.
- **pid < 0:** Estamos en el proceso padre, pero ha ocurrido un error en la ejecución de fork



# Ejercicios

1. A partir del siguiente código en C, responda:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(){
6     pid_t t = fork();
7     if(t > 0){
8         printf("Hola soy tu padre...\n");
9     }else if(t == 0){
10        printf("Hola parente\n");
11    }else{
12        printf("Oh no... Hubo un error");
13    }
14    return 0;
15 }
```

- ¿Cuantos procesos se crean?
- ¿Cuales serán las posibles salidas?



# Ejercicios

## 2. A partir del siguiente código en C, responda:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(){
6     pid_t a = fork();
7     pid_t b = fork();
8     if(a > 0 && b > 0){
9         printf("Victor\n");
10    }else if(a == 0 && b > 0){
11        printf("Yerko\n");
12    }else if(a > 0 && b == 0){
13        printf("Diego\n");
14    }else if(a == 0 && b == 0){
15        printf("Dante\n");
16    }else{
17        printf("Vicente\n");
18    }
19    return 0;
20 }
```

- ¿Cuantos procesos se crean?
- ¿Cuales serán las posibles salidas?



# Ejercicios

3. A partir de los siguientes código en C, responda:

## Código A

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(){
6     for(int i = 0 ; i < 3 ; i++){
7         pid_t t = fork();
8         if(t > 0){
9             break;
10        }
11    }
12    return 0;
13 }
```

## Código B

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(){
6     for(int i = 0 ; i < 3 ; i++){
7         pid_t t = fork();
8         if(t == 0){
9             break;
10        }
11    }
12    return 0;
13 }
```

- ¿Cuál es la principal diferencia?
- ¿Cuantos procesos se crean?
- ¿Cuales serán las posibles salidas?

# Ejercicios

## 4. A partir del siguiente código en C, responda:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(){
6     if(fork() > 0){
7         printf("Que pasa aqui!!!\n");
8     }else{
9         printf("No se\n");
10    }
11    return 0;
12 }
```

- ¿Cuantos procesos se crean?
- ¿Cuales serán las posibles salidas?

# Ejercicios

## 5. A partir del siguiente código en C, responda:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(){
6     int num = 0;
7     if(fork() > 0){
8         for(int i = 0 ; i < 5 ; i++){
9             pid_t t = fork();
10            num++;
11            if(t > 0){
12                num = num/2;
13                break;
14            }else if(t == 0){
15                num = num*2;
16            }else{
17                break;
18            }
19        }
20    }else{
21        num = 666;
22    }
23    printf("%d\n", num);
24    return 0;
25 }
```

- ¿Cuantos procesos se crean?
- ¿Cuantas posibles salidas existen?

Nombre 3

# Ejercicios

## 6. A partir del siguiente código en C, responda:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(){
6     if(fork() && fork()){
7         printf("Que esta pasando aqui!!\n");
8     }else{
9         printf("Hola, soy %d\n", getpid());
10    }
11 }
```

- ¿Cuantos procesos se crean?
- ¿Cuales serán las posibles salidas?